Quantifying Literature's Quality

New Mexico
Supercomputing Challenge
Final Report
April 1, 2015

Team 57
Los Alamos High School

Team Members:
Tabitha Welch

Teacher:
Adam Drew

Project Mentor:
Paul Welch

Abstract

The purpose of this project was to determine if there may be some correlation between a work of literature's syntax complexity and its success or popularity. Syntax complexity measures that were studied included author vocabulary size, usage of various parts of speech, frequency of various conjunctions in the text, and frequency of various punctuation marks in the text. Over the course of the year, investigations expanded to incorporate potential correlations between these metrics themselves. Analysis tools included but were not limited to basic frequency analysis, parts of speech tagging, principal component analysis, and the creation of two-dimensional histograms. All of these analyses were carried out in the R, Perl, or Java programming languages. Ultimately, results indicated that there is no correlation between syntax complexity and book popularity. However, there are several correlations between complexity metrics. These include linear relationships between total word count and numbers of different parts of speech, as well as a definitive decaying exponential relationship between author vocabulary size and book length. This project will very likely be continued in the future, but subsequent research will not focus on literature's popularity. It may instead revolve around further parts of speech analysis and comparison of English literature to literature in foreign languages.

Problem

The original goal of this project was to discover possible correlations between a work of literature's syntactical complexity and its popularity. I hypothesized that in general, a higher syntactical complexity would result in higher popularity. Usually, a book with varied word use and sentence structure is more engaging than a book with limited vocabulary and repetitive sentence structure. However, I also hypothesized that there was a limit to this trend; an extremely complex book would be difficult to understand and therefore less popular. Naturally, characterization and plot elements play a large role in literature's popularity as well. This year, I studied four major syntax complexity metrics.

These were the frequency of conjunctions in the text, the frequency of various other parts of speech in the text, the frequency of various punctuation marks in the text, and the size of the author's vocabulary, i.e. how varied their word usage was. I also examined the potential exponential relationship between the author's vocabulary size and the number of sentences in a work. This year's data set consisted of 199 fiction works which were downloaded from Project Gutenberg, an online source of literature in the public domain. The samples equally represented all genres, including short stories as well as novels. Gutenberg also provided the number of times each sample had been downloaded in the past month; this served as my popularity metric throughout the project.

Methods

Frequency Analysis

The first analysis I performed this year was an extremely simple frequency analysis on punctuation and conjunctions. I created a class in Java that had methods intended to count various punctuation marks or conjunctions in a given text (see Appendix 1). The conjunction frequency method searched for 25 specific conjunctions; the full list can be viewed in Appendix 1. Of those conjunctions, the words “and”, “but”, “or”, and “because” were specifically counted, and the rest were grouped in a category called “other.” A total conjunction count based on these categories was also calculated and returned. Importantly, a word could only be considered a conjunction if it occurred directly after a comma. Therefore, the “and” in “he danced, and he sang” would be counted as a conjunction, but the “and” in “bread and butter” would not. The punctuation frequency method functioned on a similar algorithm, but it read each file character-by-character instead of word-by-word. The goal of this method was not to count all punctuation marks in a text, but rather to obtain the frequencies of specific ones. In particular, the code searched for periods, exclamation points, question marks, commas, semicolons, and colons. It was also capable of adding the number of periods, exclamation points, and question marks to determine the total number of sentences in a work. This sentence count was later useful in determining average sentence length in a given sample.

Dictionary Generator

For each book in the data set, I generated a “dictionary” file consisting of a list of all words that were used in the text. Each dictionary was accompanied by a list of frequencies, each frequency being the number of times the corresponding dictionary word appeared. Generating these files required two different codes, both written in Perl (see Appendices 2 and 3). First, the sample in question passed through a punctuation remover code. This ensured that the same words with different punctuation would not be counted as different words in the dictionary. The code read the sample one word at a time, searching for all common punctuation marks. When such a punctuation mark was found, it was split from the word. The word was then added to a new file. Words that did not contain punctuation were immediately added to this file. The process was repeated until the code produced a file with no punctuation at all. Then, this punctuation-free text could be passed to the dictionary generator itself. The generator made use of a hash table; it read the parsed text one word at a time, compared that word to words already in the hash table, and added it to the table if no match was found. The code also kept count of how many times each word appeared, and this was used to generate each dictionary's frequency file. Dictionaries and word frequencies were used to calculate author vocabulary size, total number of words, and average sentence length.

Stanford Parts of Speech Tagger

Stanford University's Natural Language Processing Group offers several parts of speech tagging models to the general public. Each model reads through a text file and assigns every word in that file a specific part of speech; for instance, the word “my” would be tagged as a possessive pronoun and the word “completes” would be tagged as a present-tense verb in third person singular (Atwell). The

English left3words model was used for this project. After tagging my data set, I wrote my own code in Java that counted the number of various tags. I obtained counts for all verbs, nouns, pronouns, adjectives, and adverbs. These counts were later used in basic scatter plots, principal component analysis, and two-dimensional histograms.

Principal Component Analysis

I used the R Principal Component Analysis tool on a group of nine metrics (average sentence length, normalized vocabulary size, normalized conjunction frequency, total number of words, and the normalized frequencies of the following parts of speech: adjectives, adverbs, verbs, nouns, and pronouns) in order to search for a less obvious correlation to popularity. Principal component analysis allowed me to determine if some combination of these nine metrics would result in a successful book, instead of a single metric holding the key. The main idea behind PCA is that each metric is like a separate dimension in a coordinate system, but these dimensions do not allow one to easily see patterns. PCA uses matrices to rotate the coordinate system and create new dimensions or metrics (called principal components), which are combinations of the old ones and which might reveal patterns more easily. I eventually plotted each of my nine principal components with the books' popularities.

Two-Dimensional Histogram Analysis

Finally, I used R's hexbin library to create two-dimensional histograms of my data. A two-dimensional histogram follows the same basic principle as a one-dimensional histogram, but it compares two variables rather than merely displaying the frequencies of one. Hexbin divides the graph into equal hexagonal areas and determines how many data points fall within each. Darker shades of gray on the resulting histogram indicate a greater number of data points. Two-dimensional histograms were primarily used for popularity analysis this year. I compared various ratios of parts of speech to book popularity (e.g., the ratio of nouns to adjectives vs. popularity), and I also made use of these histograms to compare various other metrics to popularity.

Model Verification

All validation of my codes was accomplished via test passages; I manually analyzed these and compared my analysis to the computer's during the debugging process. For an example of such a test passage, see Appendix 5. This passage is the slightly modified first paragraph of *Pride and Prejudice*. It is reasonably short, so I can easily perform a manual text analysis on it. This paragraph has 89 words, and enough punctuation that I can be confident that the punctuation remover and dictionary generator codes are functioning properly. As additional verification of the parts of speech tagger, I could also examine the tagged text produced from running the tagger model on this passage. This would allow me to readily observe that all tags were fitting for their corresponding words.

Results

Relationship Between Number of Conjunctions and Book Popularity

A basic scatter plot of normalized conjunction count vs. popularity reveals a potential parabolic relationship between these two metrics (see Figures 1, 2, and 3). This relationship is by no means definitive, and this type of plot may not be the best for confirming or refuting it. Naturally, a popular book depends on much more than the perfect ratio of conjunctions to other words. Nonetheless, a curve can be seen on three different levels of popularity, suggesting an ideal number of compound or compound-complex sentences (those with conjunctions). Conjunction frequency for each sample was normalized by dividing the number of conjunctions found by the total number of words in the text.

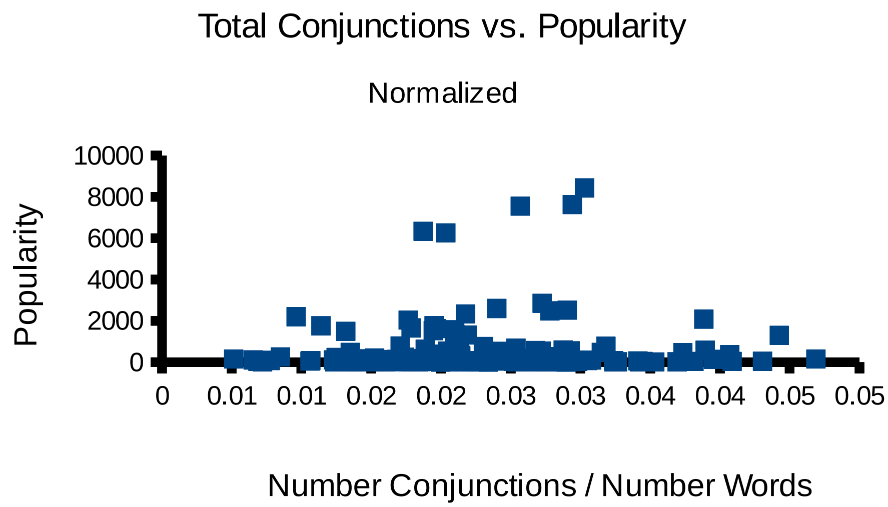


Figure 1. Scatter plot of normalized conjunction count vs. popularity. Max. y value = 10000.

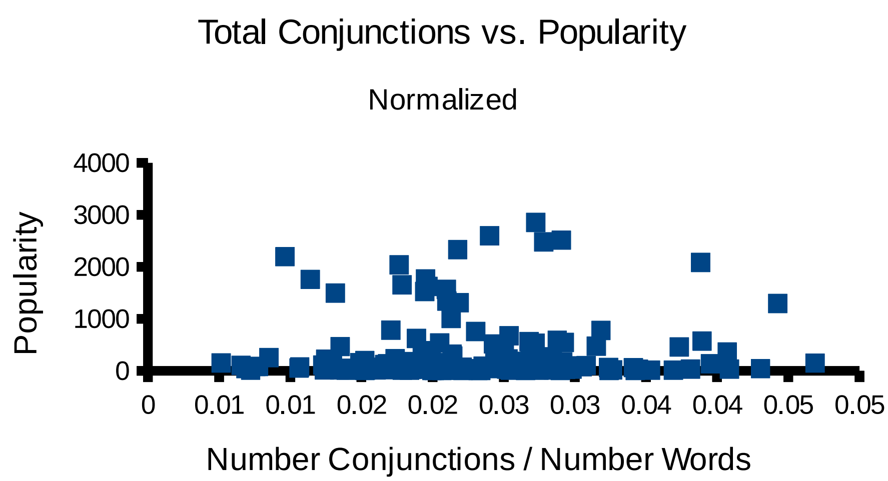


Figure 2. Scatter plot of normalized conjunction count vs. popularity. Max. y value = 4000.

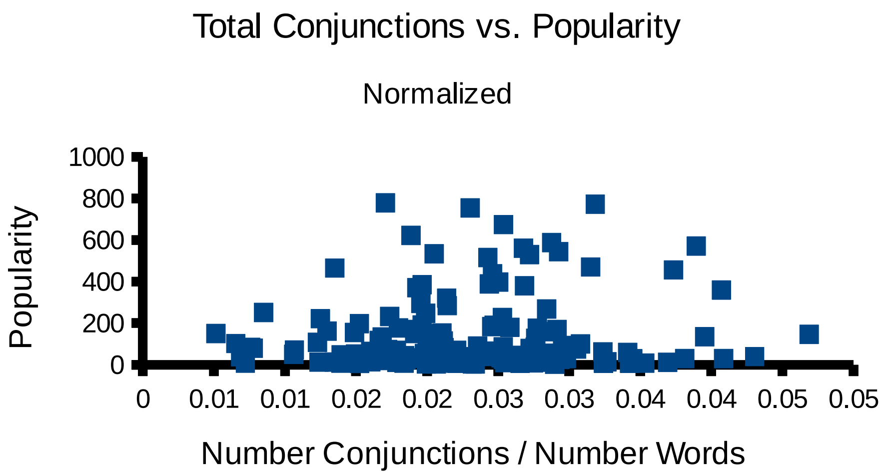


Figure 3. Scatter plot of normalized conjunction count vs. popularity. Max. y value = 1000.

Relationship Between Other Metrics and Book Popularity

Figures 4 and 5 depict two selected scatter plots of other complexity metrics vs. popularity. To all appearances, there are no relationships revealed; the plots are random. This suggests that there is no correlation between very simple metrics like book length or comma frequency and popularity. If there are any relationships at all, they are likely to only be revealed through more thorough or complex analyses.

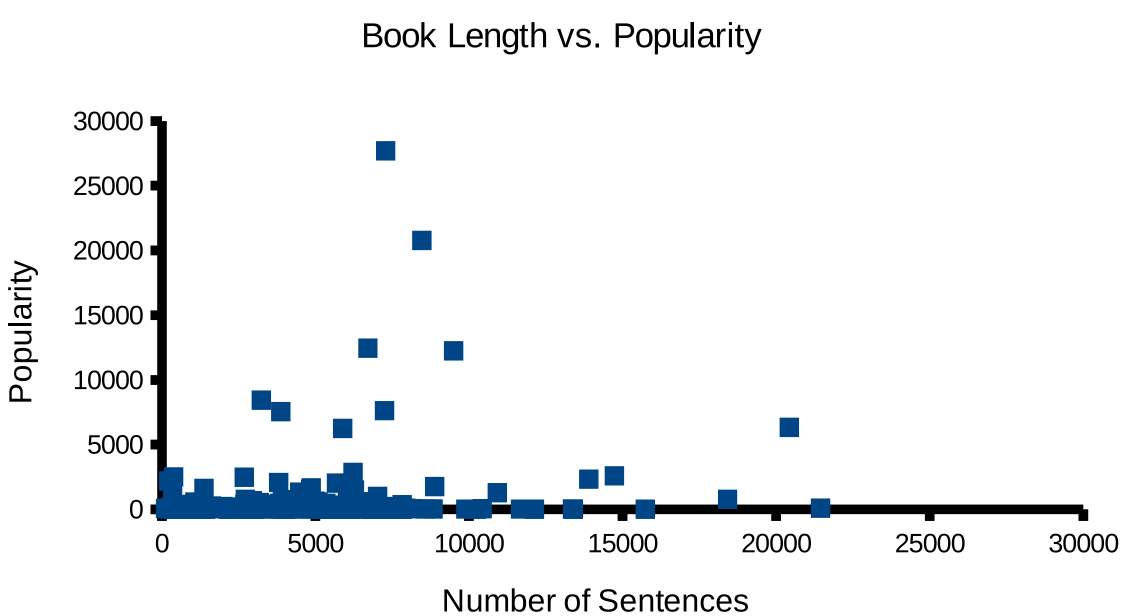


Figure 4. Scatter plot of book length (measured in sentences) vs. popularity.

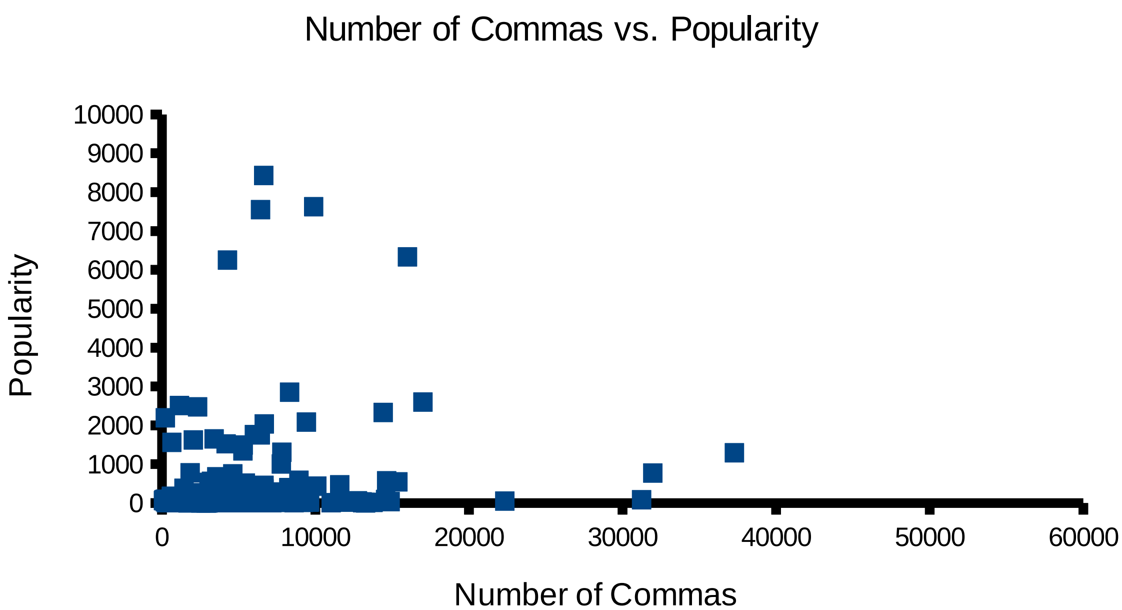


Figure 5. Scatter plot of comma frequency vs. popularity.

Principal Component Analysis and Book Popularity

The nine scatter plots in Appendix 6 depict the results of the principal component analysis when compared to popularity. Note that for scaling reasons, the y-axis is actually the logarithm of book popularity. All of these plots look extremely similar; all data points fall within a single PCA score range, whether they are very popular or practically unknown. This range is always centered around x = 0.0. The only exception to this pattern is the plot concerning principal component #2, but it is entirely random, with points scattered almost evenly across the graph. These plots show very clearly that no combination of the nine metrics on which PCA was performed will produce a successful book. In this case, principal component analysis failed to detect more subtle patterns and correlations between my metrics and popularity.

Relationship Between Vocabulary Size and Book Length

As seen in Figure 6, there is a pronounced decaying exponential relationship between author vocabulary size and the number of sentences in a work. This correlation was originally revealed in my project last year; the larger data set that I used this year supports that conclusion despite a change in my calculation of sentence count. Last year, I only considered periods as end-of-sentence indicators, but this year I modified my algorithm to count exclamation points and question marks as well. The result was a more accurate calculation. Note that author vocabulary size was normalized by dividing the number of unique words used by the total number of words used. The exponential curve shows that typically, an author's normalized vocabulary decreases drastically as the book's length increases; a book that contains many more words does not necessarily contain many more unique words. Stated differently, an author will not constantly introduce new words into his or her text but will continue to work with the same vocabulary throughout. For the exact equation of the curve, refer also to Figure 6.

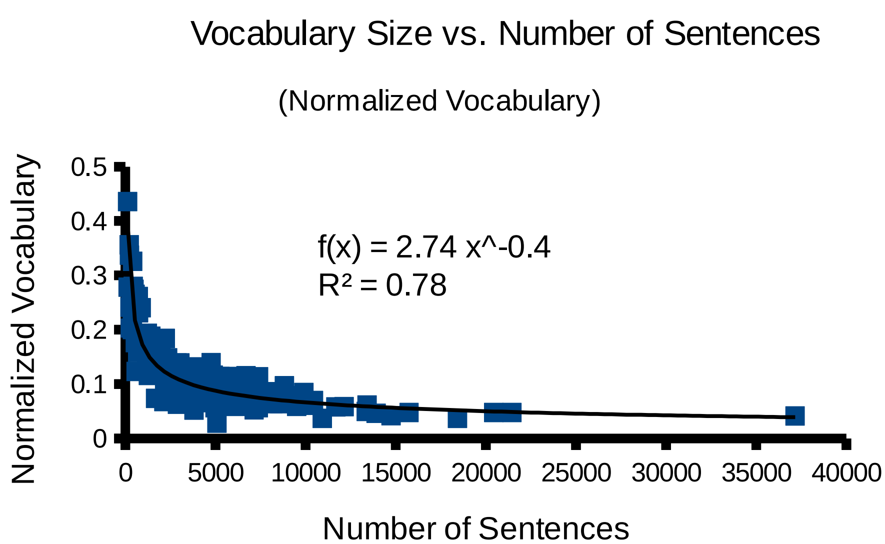


Figure 6. Scatter plot of book length (measured in sentences) vs. normalized vocabulary size. Equation describes exponential decay curve.

Two-Dimensional Histogram Analysis With Respect to Popularity

I created two-dimensional histograms comparing book popularity with a wide variety of data metrics, particularly those involving different parts of speech. To view these histograms, see Appendix 7 at the end of the report. Also with respect to popularity, the ratios of nouns to adjectives, adjectives to nouns, nouns to verbs, and verbs to nouns were plotted as two-dimensional histograms. Somewhat surprisingly, no correlation is visible between any of these ratios and book popularity. The data is randomly scattered across each histogram. This may be due to insufficient data; however, it appears that popularity is not remotely connected to the number of descriptors per noun, etc. I also generated histograms that directly compared popularity to frequencies of conjunctions, frequencies of various punctuation marks, and average sentence length (see Appendix 7). Again, no correlations are readily visible. The most popular books on the histograms always fall within the same range vertically as the least popular books. Clearly, none of these metrics are strongly related to book popularity, despite the basic scatter plots of conjunction frequency and popularity discussed above.

Scatter Plots With Respect to Parts of Speech

I also created scatter plots to compare frequencies of various parts of speech to a sample's total word count. The plots for nouns, pronouns, verbs, adjectives, and adverbs all revealed strong linear correlations to word count. This shows that for all types of books, the ratio of a particular part of speech to other words will remain fairly constant. I performed a linear regression on each of my plots for these parts of speech, and in this way I was able to determine the exact ratio. For an example, see Figure 7. According to this regression, approximately one in every 4 words will be a noun. Equations for the remaining regressions can be seen in Table 1. About one in every 5 words will be a verb, one in every 8 will be a pronoun, one in every 15 will be an adjective, and about one in every 14 words will be an adverb. For the remaining part of speech scatter plots, see Appendix 7. Note also that all linear models fit these plots very well; the average R^2 value is 0.97.

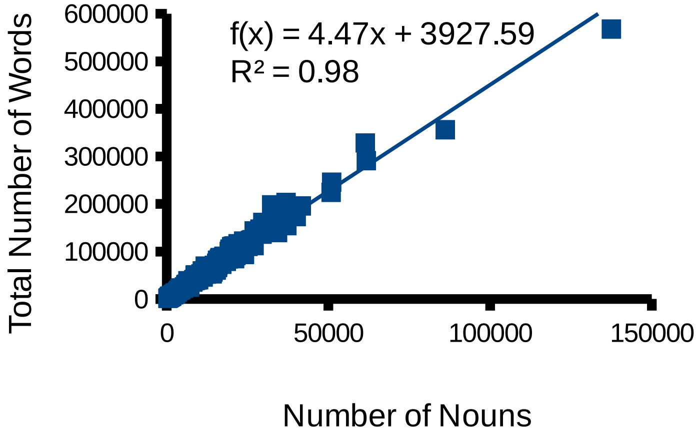


Figure 7. Scatter plot depicting the ratio of noun count to total word count.

Table 1. Regression lines for total words vs. various parts of speech.

Part of Speech	**Regression Line**
Noun	f(x) = 4.47x + 3927.59
Verb	f(x) = 5.30x – 1391.36
Pronoun	f(x) = 7.80x + 326.70
Adjective	f(x) = 14.80x + 1144.12
Adverb	f(x) = 14.30x -1798.84

Conclusions

Overall, most of my metrics showed no correlation to book popularity, with the potential exception of conjunction frequency. My results provide very strong evidence that a good book is determined not by a single “magic formula” but by the content itself; creating a good story is a matter of art, and readers will pay more attention to the author's message than how many different words he or she used.

On the other hand, I did discover correlations between different syntactical components of a book, namely the relationship between vocabulary size and book length and the nearly constant ratios of different parts of speech. The latter were certainly unexpected and intriguing results. Writers who differ vastly in matters of style and quality of production use nearly the same ratios of descriptors, action words, nouns, and so on. According to these results, someone who spends entire paragraphs describing a single scene actually utilizes about as many adjectives on average as someone who barely lays out a setting at all.

As always, there is ample room for improvement and expansion on this project. Perhaps new correlations would emerge if I used Stanford's part of speech tagger to classify words more specifically. For instance, rather than identifying the word “completes” merely as a verb, I might treat it as a present-tense verb in third person singular. Future work on the project may also include analysis of foreign languages and the comparison of usage of those languages to the usage of English. However, I will likely not continue to search for a relationship between syntax and popularity.

Significant Achievements

Many of my significant achievements this year involved my process more than they did my results. Some of the most important components of the project included understanding and using analysis tools such as principal component analysis. Particularly significant results included the discovery of set ratios for use of different parts of speech and the conclusion that truly exceptional literature is about much, much more than the most appealing syntax.

Software

I used Java and Perl for all codes this year. The dictionary and punctuation remover codes were written in Perl; all other codes were written in Java. Stanford's POS Tagger was implemented in Java as well. I used R and LibreOffice Calc for my final plots and for the principal component analysis.

My selection of coding languages was largely a matter of personal preference. The Perl codes were both codes that I originally wrote for last year's project. However, this year I have become increasingly familiar with Java and prefer it for working with text files, text scanning, and other text analysis.

References

Atwell, Eric. "The University of Pennsylvania (Penn) Treebank Tag-set." *The University of Pennsylvania (Penn) Treebank Tag-set*. Leeds University, n.d. Web. 18 Mar. 2015.

Conway, Drew, and John Myles White. *Machine Learning for Hackers*. Beijing: O'Reilly, 2012. Print.

Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. 2003. *Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network*. In *Proceedings of HLT-NAACL 2003*, pp. 252-259.

Lewis, John, and William Loftus. *Java Software Solutions: Foundations of Program Design.* Boston: Addison-Wesley, 2012. Print.

"Project Gutenberg." *Project Gutenberg*. N.p., 11 Oct. 2014. Web. 06 Dec. 2014.

"The Stanford NLP (Natural Language Processing) Group." *The Stanford NLP (Natural Language Processing) Group*. Stanford University, 29 Jan. 2015. Web. 09 Mar. 2015.

Acknowledgements

The Stanford University Natural Language Processing Group's website provided the parts of speech tagger download (the English left3words model) that was used for this project. I would like to thank the Supercomputing Challenge judges and other staff for their support and advice throughout this year.

Additional Plots, Tables, and Codes

Table 2. List of All Data Books

A Houseful of Girls
A Love Episode
A Mountain Woman
A Red Wallflower
A Tale of Two Cities
Alarm Clock
Alive In The Jungle
An Open - Eyed Conspiracy: An Idyl of Saratoga
Bred in the Bone
Buffalo Bill's Spy Trailer
Cast Adrift
Chanticleer
Crome Yellow
David Dunne
Dora Deane, Or, The East India Uncle
Elsie at Home
Expediter
Fame and Fortune
Flames
Gabriel and the Hour Book
How It All Came Round
Huckleberry Finn
In Her Own Right
In The Brooding Wild
Instinct
Judith of the Plains
King Spruce
Les Miserables
Little Lost Sister
Little Mittens for the Little Darlings
Love and Mr. Lewisham
Mary Gray
My Fair Planet
Ned Garth
Old Man Curry: Race Track Stories
Operation Earthworm
Peveril of the Peak
Polly Oliver's Problem
Poor Jack
Precaution: A Novel
Pride and Prejudice
Rewards and Fairies

Squinty the Comical Pig
Sunny Boy in the Country
Sustained Honor
Swiss Family Robinson
That Affair Next Door
The Adventure Club Afloat
The Adventures of Harry Revel
The Adventures of Sherlock Holmes
The Black Bag
The Captives
The Chums of Scranton High
The Dope on Mars
The Dragon of Wantley
The Efficiency Expert
The Eye of Zeitoon
The Goat and Her Kid
The Good Neighbors
The Grizzly King
The Home in the Valley
The Hunters
The Inhabited
The Inner Sisterhood
The Invader
The Iron Woman
The Island of Faith
The King in Yellow
The Lost City
The Mayor of Casterbridge
The Mystery of the Four Fingers
The Old Folks' Party
The Patchwork Girl of Oz
The Prince and the Page: A Story of the Last Crusade
The Puppet Crown
The Radio Boys on the Mexican Border
The Red House Mystery
The Rover Boys At College
The Severed Hand
The Silver Butterfly
The Slave of Silence
The Spinners
The Spinster
The Splendid Folly
The Story of Red Feather: A Tale of the American Frontier
The Sword Maker
The Tragedy of the Chain Pier
The Turmoil: A Novel

The Valley of Decision
The Witness
The Young Lieutenant
Wanted - 7 Fearless Engineers!
Wayside Courtships
We Didn't Do Anything Wrong, Hardly
Wessex Tales
With Airship and Submarine
With Wolfe in Canada
Wolves of the Sea
Woman Triumphant
Wood Magic: A Fable
Bones
Hunter Quatermain's Story
Tarzan of the Apes
The Scarlet Pimpernel
The Prisoner of Zenda
The Thirty-Nine Steps
The Black Arrow: A Tale of Two Roses
The Return of Dr. Fu-Manchu
Mr. Justice Raffles
The Extraordinary Adventures of Arsene Lupin, Gentleman-Burglar
The Golden Scorpion
An African Millionare: Episodes in the Life of the Illustrious Colonel Clay
Crime and Punishment
Murder in the Gunroom
Within an Inch of His Life
The Moon Rock
The Ashiel Mystery: A Detective Story
The Mysterious Affair at Styles
Dead Men's Money
The Moonstone
In the Fog
The Merry Adventures of Robin Hood
The Book of Wonder
The Crock of Gold
Gulliver of Mars
The Wood Beyond the World
The Legends of King Arthur and His Knights
The Night Land
The Yellow Wallpaper
The Phantom of the Opera
The History of Caliph Vathek
Wuthering Heights
The Vampyre; A Tale
The Mysteries of Udolpho

The Picture of Dorian Gray
A Thane of Wessex
Grisly Grisell; Or, the Laidly Lady of Whitburn: A Tale of the Wars of the Roses
By Pike and Dyke: a Tale of the Rise of the Dutch Republic
The Pilot: A Tale of the Sea
Treasure Island
The Virginians
A Gentleman of France: Being the Memoirs of Gaston de Bonne Sieur de Marsac
A Thin Ghost and Others
Four Weird Tales
Dracula's Guest
An Occurrence at Owl Creek Bridge
The House of Souls
Varney the Vampire; Or, the Feast of Blood
The Shunned House
The Diary of a Nobody
My Man Jeeves
Samantha at Saratoga
A Prefect's Uncle
Miss Mapp
Comic History of England
In Brief Authority
The Age of Innocence
The Auction Block
A Journey to the Centre of the Earth
The Scarlet Letter
The First Men in the Moon
Moonfleet
"Captains Courageous": A Story of the Grand Banks
The Secret Agent: A Simple Tale
The Woman in White
The Beetle: A Mystery
Greenmantle
Dracula
The Mystery of Edwin Drood
The Last Man
A Connecticut Yankee in King Arthur's Court
The Dominion in 1983
Trips to the Moon
A Crystal Age
The Strange Case of Dr. Jekyll and Mr. Hyde
The Diamond Lens
The Boys of Bellwood School; Or, Frank Jordan's Triumph
For the Sake of the School
Daddy-Long-Legs
A Little Princess

The White Feather
Glyn Severn's Schooldays
What Katy Did At School
The Sensitive Man
Under Arctic Ice
Pandemic
The Lost Continent
Badge of Infamy
The Barbarians
The Revolt of the Star Men
Bucky O'Connor: A Tale of the Unfenced Border
Black Jack
The Barrier
The Hidden Children
Rebel Spurs
Betty Zane
Cow-Country

Appendix 1. The TextObject Class (Java Code).

```
//This is a class that will have various methods to analyze the text.

import java.io.File;
import java.util.Scanner;
import java.io.BufferedReader;
import java.io.FileReader;

public class TextObject{

//basic constructor
public TextObject(){

}

//This code counts: that, as, if, when, than, and, or, but, because, while, where,
      //after,so, though, since, until, whether, before, although, nor, like, once,
      //unless, now, except
public int JavaConjunctions(File textfile) throws Exception{

      File file = textfile;
      Scanner fileScan = new Scanner(file);
      int andcount = 0, butcount = 0, orcount = 0, bccount = 0, othercount = 0;

      while(fileScan.hasNext()){

            String wordbefore = fileScan.next();
            char lastchar = wordbefore.charAt(wordbefore.length() - 1);
            if(lastchar == 44){    //44 = ASCII for ,

                  String aword = fileScan.next();

                  if(aword.equals("and") || aword.equals("\"and") ||
                     aword.equals("and,")){
                        andcount++;
                  } else {
                        if(aword.equals("but") || aword.equals("\"but") ||
                            aword.equals("but,")){
                              butcount++;
                        } else {
                            if(aword.equals("or") || aword.equals("\"or") ||
                                 aword.equals("or,")){
                                    orcount++;
                              } else {
                               if(aword.equals("because") ||
                                    aword.equals("\"because") ||
                                    aword.equals("because,")){
                                          bccount++;
                                    }
                              }
                        }
                  }

                  if(aword.equals("that") || aword.equals("\"that") ||
                    aword.equals("that,")){
```

```
    othercount++;
}
if(aword.equals("as") || aword.equals("\"as") ||
  aword.equals("as,")){
    othercount++;
}
if(aword.equals("if") || aword.equals("\"if") ||
   aword.equals("if,")){
    othercount++;
}
if(aword.equals("when") || aword.equals("\"when") ||
   aword.equals("when,")){
    othercount++;
}
if(aword.equals("than") || aword.equals("\"than") ||
  aword.equals("than,")){
    othercount++;
}
if(aword.equals("while") || aword.equals("\"while") ||
  aword.equals("while,")){
    othercount++;
}
if(aword.equals("where") || aword.equals("\"where") ||
  aword.equals("where,")){
    othercount++;
}
if(aword.equals("after") || aword.equals("\"after") ||
  aword.equals("after,")){
    othercount++;
}
if(aword.equals("so") || aword.equals("\"so") ||
  aword.equals("so,")){
    othercount++;
}
if(aword.equals("though") || aword.equals("\"though") ||
  aword.equals("though,")){
    othercount++;
}
if(aword.equals("since") || aword.equals("\"since") ||
   aword.equals("since,")){
    othercount++;
}
if(aword.equals("until") || aword.equals("\"until") ||
   aword.equals("until,")){
    othercount++;
}
if(aword.equals("whether") || aword.equals("\"whether") ||
   aword.equals("whether,")){
    othercount++;
}
if(aword.equals("before") || aword.equals("\"before") ||
   aword.equals("before,")){
    othercount++;
}
if(aword.equals("although") || aword.equals("\"although") ||
  aword.equals("although,")){
    othercount++;
```

```
                }
                if(aword.equals("nor") || aword.equals("\"nor") ||
                  aword.equals("nor,")){
                    othercount++;
                }
                if(aword.equals("like") || aword.equals("\"like") ||
                  aword.equals("like,")){
                    othercount++;
                }
                if(aword.equals("once") || aword.equals("\"once") ||
                  aword.equals("once,")){
                    othercount++;
                }
                if(aword.equals("unless") || aword.equals("\"unless") ||
                  aword.equals("unless,")){
                    othercount++;
                }
                if(aword.equals("now") || aword.equals("\"now") ||
                  aword.equals("now,")){
                    othercount++;
                }
                if(aword.equals("except") || aword.equals("\"except") ||
                  aword.equals("except,")){
                    othercount++;
                }
                 }
        }

        int totalcount = andcount + butcount + orcount + bccount + othercount;

        return andcount;
        //return butcount;
        //return orcount;
        //return bccount;
        //return totalcount;

}

//Counts numbers of various punctuation and then calculates the number of
sentences.
public int JavaPunctuationCounter(File textfile) throws Exception{

        BufferedReader textfileReader = new BufferedReader(new FileReader(textfile));
        int asciichar = textfileReader.read();
        int periodcount = 0, exclamationcount = 0, questioncount = 0, commacount = 0;
        int semicoloncount = 0, coloncount = 0, doublequotecount = 0;

        while(asciichar != -1){
                char thischar = (char)asciichar;

                if(thischar == '.'){
                        periodcount++;
                }
                if(thischar == '?'){
                        questioncount++;
                }
                if(thischar == '!'){
```

```
                exclamationcount++;
            }
            if(thischar == ','){
                commacount++;
            }
            if(thischar == ';'){
                semicoloncount++;
            }
            if(thischar == ':'){
                coloncount++;
            }
            if(thischar == '"'){
                doublequotecount++;
            }

            asciichar = textfileReader.read();
        }

        int numbersentences = periodcount + exclamationcount + questioncount;

        return periodcount;
        //return questioncount;
        //return exclamationcount;
        //return commacount;
        //return semicoloncount;
        //return numbersentences;

        }
}
```

Appendix 2. Punctuation Remover (Perl Code).

```
#!/usr/bin/perl

open(ifile,$ARGV[0]);
$fname = "$ARGV[0]".".parsed";
open(ofile, ">$fname");

$pcount = 0;
$commacount = 0;
$exclamationcount = 0;
$semicoloncount = 0;
$questioncount = 0;
$coloncount = 0;
$i = 0;
$w = 0;

$somepunk = 0;
while(<ifile>){
  chomp();
  @m = split(" ", $_);
  $n = @m;
  $w = $n + $w;

  #Each if statement is responsible for removing one kind of punctuation. Some
      #punctuation marks also have counters associated with them.
  for($i=0; $i<$n; $i++){
    $tagged =0;
    if($m[$i] =~ "\\."){
      $pcount++;
      @tword=split("\\.", $m[$i]);
      print "$m[$i] => $tword[0]\n";
      $tagged = 1;
      print ofile "$tword[0]\n";
    }
    if(($m[$i] =~ "\\,")&&(!$tagged)){
      $commacount++;
      @tcomma=split("\\,", $m[$i]);
      print "$m[$i] => $tcomma[0]\n";
      $tagged = 1;
      print ofile "$tcomma[0]\n";
    }
    if(($m[$i] =~ "\\!")&&(!$tagged)){
      $exclamationcount++;
      @texclamation=split("\\!", $m[$i]);
      print "$m[$i] => $texclamation[0]\n";
      $tagged = 1;
      print ofile "$texclamation[0]\n";
    }
    if(($m[$i] =~ "\/")&&(!$tagged)){
      @texclamation=split("\/", $m[$i]);
      print "$m[$i] => $texclamation[0]\n";
      $tagged = 1;
      print ofile "$texclamation[0]\n";
    }
    if(($m[$i] =~ /\\/)&&(!$tagged)){
```

```
    @tbslash=split(/\\/, $m[$i]);
    print "$m[$i] => $tbslash[0]\n";
    $tagged = 1;
    print ofile "$tbslash[0]\n";
  }
  if(($m[$i] =~ "\\;")&&(!$tagged)){
    $semicoloncount++;
    @tsemicolon=split("\\;", $m[$i]);
    print "$m[$i] => $tsemicolon[0]\n";
    $tagged = 1;
    print ofile "$tsemicolon[0]\n";
  }
  if(($m[$i] =~ "\\:")&&(!$tagged)){
    $coloncount++;
    @tcolon=split("\\:", $m[$i]);
    print "$m[$i] => $tcolon[0]\n";
    $tagged = 1;
    print ofile "$tcolon[0]\n";
  }
  if(($m[$i] =~ "\\_")&&(!$tagged)){
    @tsquote=split("_", $m[$i]);
    if(@tsquote >1){
      print "$m[$i] => $tsquote[1]\n";
      print ofile "$tsquote[1]\n";
    }else{print "$m[$i] => $tsquote[0]\n";
      print ofile "$tsquote[0]\n";}
      $tagged = 1;
  }
  if(($m[$i] =~ "\\*")&&(!$tagged)){
    @tsquote=split("\\*", $m[$i]);
    if(@tsquote >1){
      print "$m[$i] => $tsquote[1]\n";
      print ofile "$tsquote[1]\n";
    }else{print "$m[$i] => $tsquote[0]\n";
      print ofile "$tsquote[0]\n";}
      $tagged = 1;
  }
  if(($m[$i] =~ "\\?")&&(!$tagged)){
    $questioncount++;
    @tquestion=split("\\?", $m[$i]);
    print "$m[$i] => $tquestion[0]\n";
    $tagged = 1;
    print ofile "$tquestion[0]\n";
  }
  if(($m[$i] =~ /"/ )&&(!$tagged)){
     @tsquote=split( /"/, $m[$i]);
     if(@tsquote >1){
      print "$m[$i] => $tsquote[1]\n";
      print ofile "$tsquote[1]\n";
    }else{print "$m[$i] => $tsquote[0]\n";
      print ofile "$tsquote[0]\n";}
      $tagged = 1;
  }
  if(($m[$i] =~ "\\[")&&(!$tagged)){
    @tobracket=split("\\[", $m[$i]);
    print "$m[$i] => $tobracket[1]\n";
    $tagged = 1;
```

```
  print ofile "$tobracket[1]\n";
}
if(($m[$i] =~ "\\<")&&(!$tagged)){
  @tobracket=split("<", $m[$i]);
  print "$m[$i] => $tobracket[1]\n";
  $tagged = 1;
  print ofile "$tobracket[1]\n";
}
if(($m[$i] =~ "\\(")&&(!$tagged)){
  @tobracket=split("\\(", $m[$i]);
  print "$m[$i] => $tobracket[1]\n";
  $tagged = 1;
  print ofile "$tobracket[1]\n";
}
if(($m[$i] =~ "\\{")&&(!$tagged)){
  @tobracket=split("\\{", $m[$i]);
  print "$m[$i] => $tobracket[1]\n";
  $tagged = 1;
  print ofile "$tobracket[1]\n";
}
if(($m[$i] =~ "\\]")&&(!$tagged)){
  @tobracket=split("\\]", $m[$i]);
  print "$m[$i] => $tobracket[0]\n";
  $tagged = 1;
  print ofile "$tobracket[0]\n";
}
if(($m[$i] =~ "\\>")&&(!$tagged)){
  @tobracket=split(">", $m[$i]);
  print "$m[$i] => $tobracket[0]\n";
  $tagged = 1;
  print ofile "$tobracket[0]\n";
}
if(($m[$i] =~ "\\)")&&(!$tagged)){
  @tobracket=split("\\)", $m[$i]);
  print "$m[$i] => $tobracket[0]\n";
  $tagged = 1;
  print ofile "$tobracket[0]\n";
}
if(($m[$i] =~ "\\}")&&(!$tagged)){
  @tobracket=split("\\}", $m[$i]);
  print "$m[$i] => $tobracket[0]\n";
  $tagged = 1;
  print ofile "$tobracket[0]\n";
}
if(($m[$i] =~ "\\~")&&(!$tagged)){
  @tobracket=split("\\~", $m[$i]);
  print "$m[$i] => $tobracket[0]\n";
  $tagged = 1;
  print ofile "$tobracket[0]\n";
}
if(($m[$i] =~ "\\`")&&(!$tagged)){
  @tobracket=split("\\`", $m[$i]);
  print "$m[$i] => $tobracket[0]\n";
  $tagged = 1;
  print ofile "$tobracket[0]\n";
}
if(($m[$i] =~ "\\&")&&(!$tagged)){
```

```
    @tobracket=split("\\&", $m[$i]);
    print "$m[$i] => $tobracket[0]\n";
    $tagged = 1;
    print ofile "$tobracket[0]\n";
  }
  if(($m[$i] =~ "\\%")&&(!$tagged)){
    @tobracket=split("\\%", $m[$i]);
    print "$m[$i] => $tobracket[0]\n";
    $tagged = 1;
    print ofile "$tobracket[0]\n";
  }
  if(($m[$i] =~ "\\+")&&(!$tagged)){
    @tobracket=split("\\+", $m[$i]);
    print "$m[$i] => $tobracket[0]\n";
    $tagged = 1;
    print ofile "$tobracket[0]\n";
  }
  if(($m[$i] =~ "\\-")&&(!$tagged)){
    @tobracket=split("\\-", $m[$i]);
    print "$m[$i] => $tobracket[0]\n";
    $tagged = 1;
    print ofile "$tobracket[0]\n";
    print ofile "$tobracket[1]\n";
  }
  if(($m[$i] =~ "\\=")&&(!$tagged)){
    @tobracket=split("\\=", $m[$i]);
    print "$m[$i] => $tobracket[0]\n";
    $tagged = 1;
    print ofile "$tobracket[0]\n";
  }
  if(($m[$i] =~ "\\@")&&(!$tagged)){
    @tsquote=split("@", $m[$i]);
    if(@tsquote >1){
      print "$m[$i] => $tsquote[1]\n";
      print ofile "$tsquote[1]\n";
    }else{print "$m[$i] => $tsquote[0]\n";
      print ofile "$tsquote[0]\n";}
      $tagged = 1;
  }
  if(($m[$i] =~ "\\^")&&(!$tagged)){
    @tsquote=split("\\^", $m[$i]);
    if(@tsquote >1){
      print "$m[$i] => $tsquote[1]\n";
      print ofile "$tsquote[1]\n";
    }else{print "$m[$i] => $tsquote[0]\n";
      print ofile "$tsquote[0]\n";}
      $tagged = 1;
  }
  if(($m[$i] =~ "\\|")&&(!$tagged)){
    @tsquote=split("\\|", $m[$i]);
    if(@tsquote >1){
      print "$m[$i] => $tsquote[1]\n";
      print ofile "$tsquote[1]\n";
    }else{print "$m[$i] => $tsquote[0]\n";
      print ofile "$tsquote[0]\n";}
      $tagged = 1;
  }
```

```
    if(!$tagged){
      print ofile "$m[$i]\n";
      print "$m[$i]\n";
    }
    if($tagged) {$somepunk = 1;}
  }
}
close(ifile);
close(ofile);

print "SomePunk = $somepunk\n";
print "Number of Periods = $pcount\n";
print "Number of Commas = $commacount\n";
print "Number of Exclamation Points = $exclamationcount\n";
print "Number of Semicolons = $semicoloncount\n";
print "Number of Colons = $coloncount\n";
print "Number of Question Marks = $questioncount\n";
```

Appendix 3. Dictionary Generator (Perl Code).

```
#!/usr/bin/perl

$keys = 0;
open(ifile, "$ARGV[0]");

#Code creates a hash table of unique words found in a text. First makes all text
      #lowercase to avoid confusion caused by capitals.
while(<ifile>){
      chomp;
      $thisword = lc($_);
      $words{$thisword}++;
      if($words{$thisword}<2){$mykey[$keys] = "$thisword"; $keys++}
}
close(ifile);

open(ofile, ">dictionary.out");
open(afile, ">frequency.out");
print ofile "I got $keys different words\n";

for($w=0; $w<$keys; $w++){
      print ofile "$mykey[$w]\n";
      print afile "$words{$mykey[$w]}\n";
}

close ofile;
close afile;
```

Appendix 4. Performing Principal Component Analysis (R Code).

```
bookdata <- read.csv('/home/tabitha/Supercomputing/PCAData6.csv')
mydat=matrix(nrow=199,ncol=9)
j = 1

#Shifts and normalizes the data.
shift1=min(bookdata[,1])
scale1=max(bookdata[,1])-shift1
shift2=min(bookdata[,2])
scale2=max(bookdata[,2])-shift2
shift3=min(bookdata[,3])
scale3=max(bookdata[,3])-shift3
shift4=min(bookdata[,4])
scale4=max(bookdata[,4])-shift4
shift5=min((bookdata[,5]/bookdata[,4]))
scale5=max((bookdata[,5]/bookdata[,4]))-shift5
shift6=min((bookdata[,6]/bookdata[,4]))
scale6=max((bookdata[,6]/bookdata[,4]))-shift6
shift7=min((bookdata[,7]/bookdata[,4]))
scale7=max((bookdata[,7]/bookdata[,4]))-shift7
shift8=min((bookdata[,8]/bookdata[,4]))
scale8=max((bookdata[,8]/bookdata[,4]))-shift8
shift9=min((bookdata[,9]/bookdata[,4]))
scale9=max((bookdata[,9]/bookdata[,4]))-shift9

#Transferring data into a data matrix.
for(i in 1:199){
  mydat[j,1] = (bookdata[i,1]-shift1)/scale1
  mydat[j,2] = (bookdata[i,2]-shift2)/scale2
  mydat[j,3] = (bookdata[i,3]-shift3)/scale3
  mydat[j,4] = (bookdata[i,4]-shift4)/scale4
  mydat[j,5] = ((bookdata[i,5]/bookdata[i,4])-shift5)/scale5
  mydat[j,6] = ((bookdata[i,6]/bookdata[i,4])-shift6)/scale6
  mydat[j,7] = ((bookdata[i,7]/bookdata[i,4])-shift7)/scale7
  mydat[j,8] = ((bookdata[i,8]/bookdata[i,4])-shift8)/scale8
  mydat[j,9] = ((bookdata[i,9]/bookdata[i,4])-shift9)/scale9
  j=j+1
}

#PCA itself.
mypca=princomp(mydat)

ranks <- read.csv('/home/tabitha/Supercomputing/Ranks.csv')

plot(mypca$scores[,1],ranks[,1])
```

Appendix 5. Test Passage For Model Verification.

{PRIDE AND` PREJUDICE}

By Jane} Austen]

Chapter% 1)

It_. it is a truth universally~ acknowledged, that^ a ^single man in possession
_of_ a |good| fortune, must be in want of a wife.

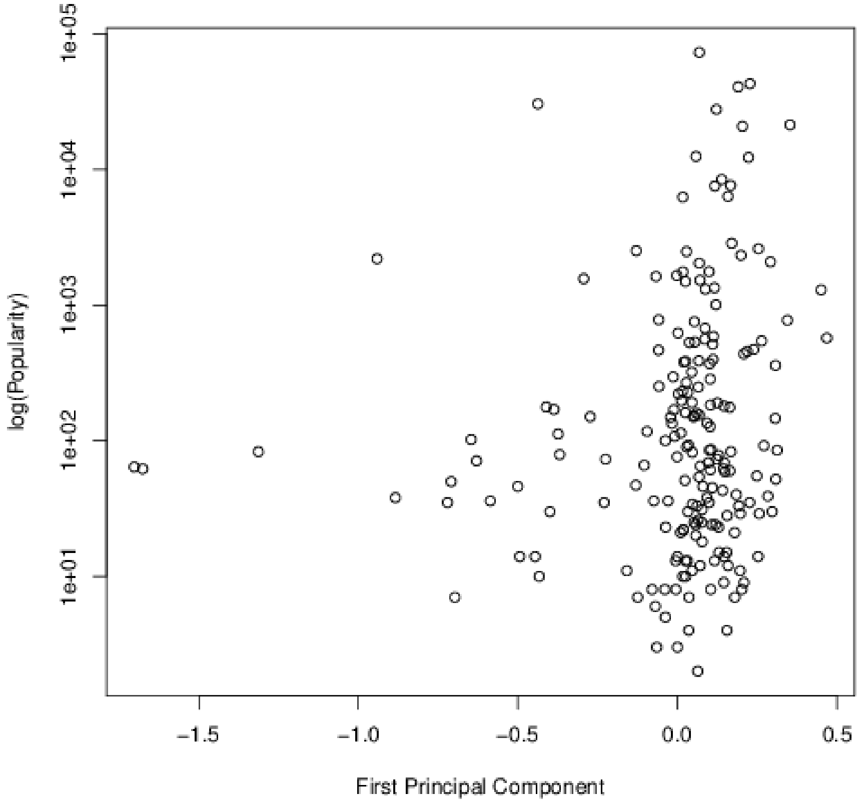
"However". little = known the feelings or views of - such a man may be on his
@first entering@ a neighbourhood, this truth is-so well fixed in <the minds
of* the *surrounding families, that he is considered the rightful> <property>
of some one + or other of; their & daughters.

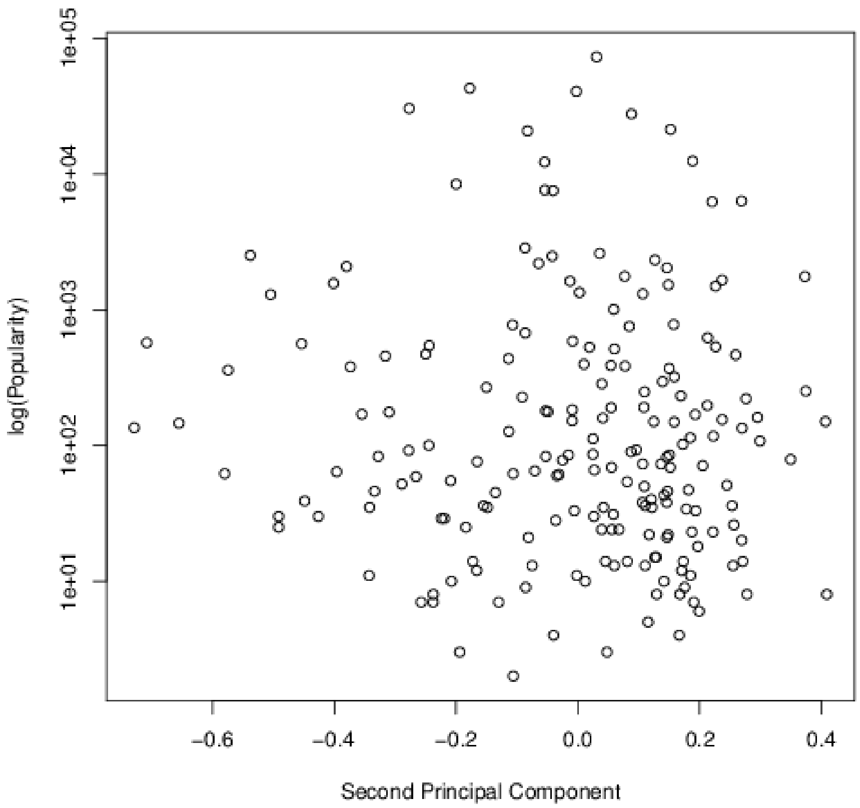
Bob.

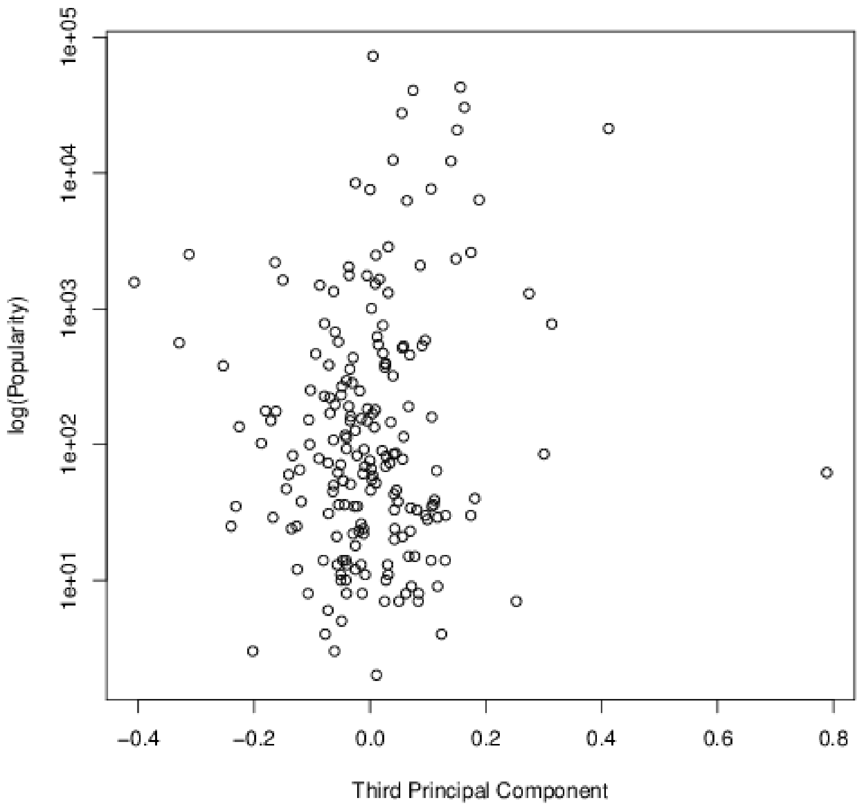
[testing]
(just: in case)

? ?

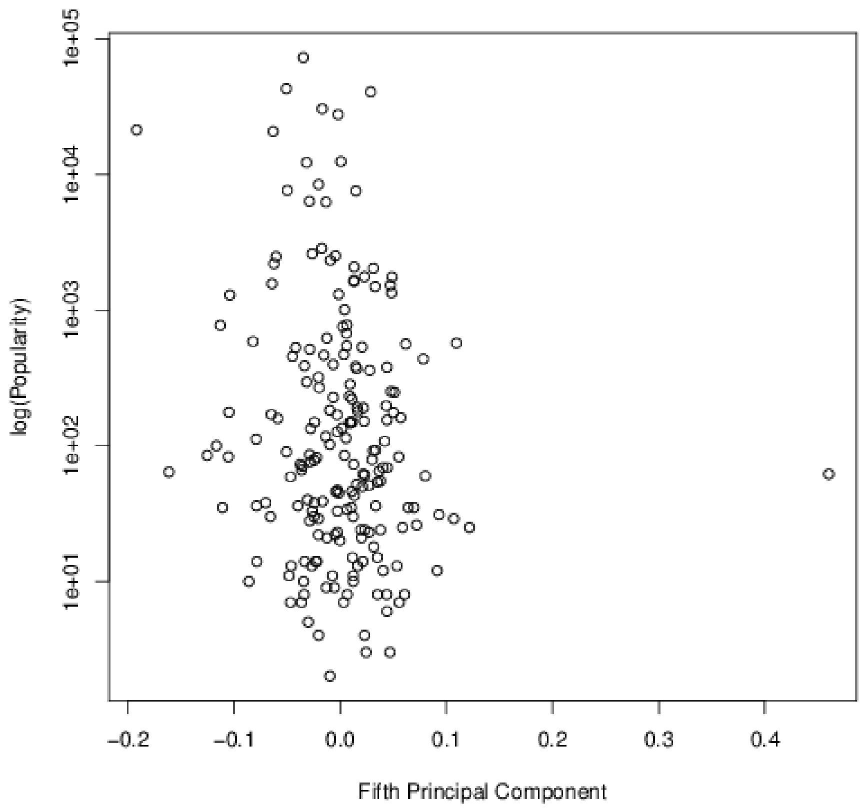
Appendix 6. Principal Component Analysis Results. See Also Appendix 4.



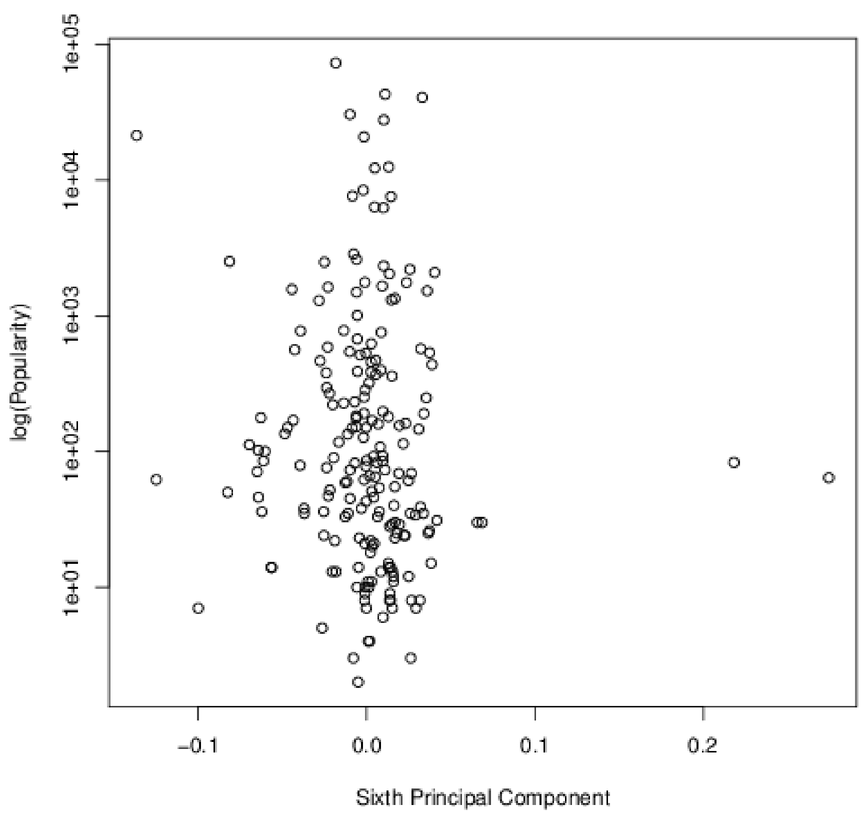



1e+05
1e+04
1e+03
1e+02
1e+01
log(Popularity)
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
Third Principal Component

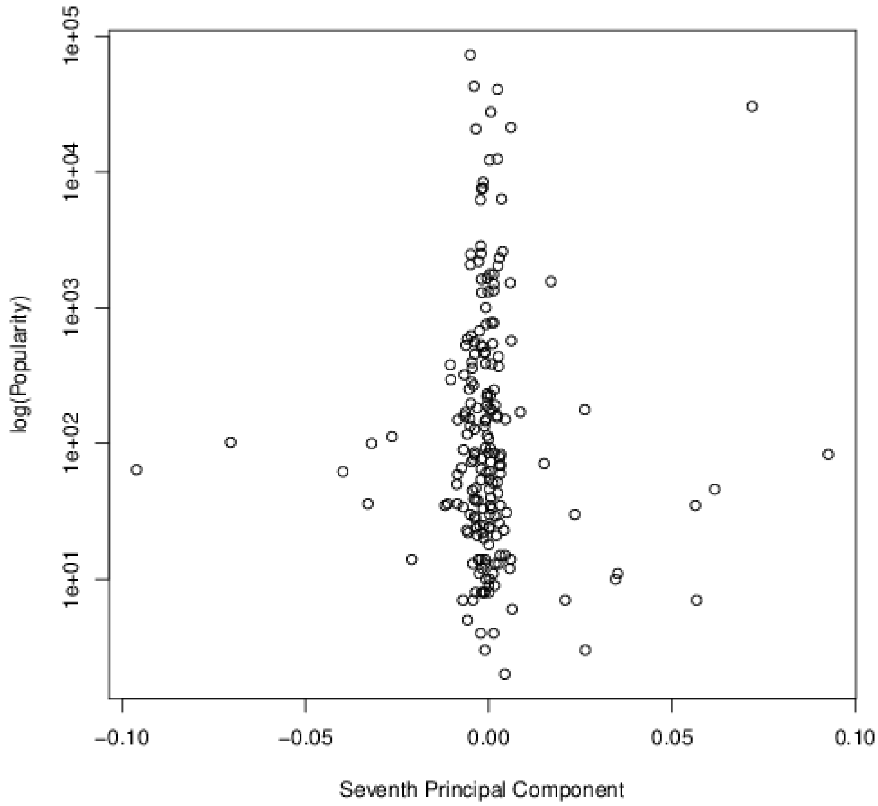

1e+05
1e+04
1e+03
1e+02
1e+01
log(Popularity)
−0.6
−0.4
−0.2
0.0
0.2
Fourth Principal Component

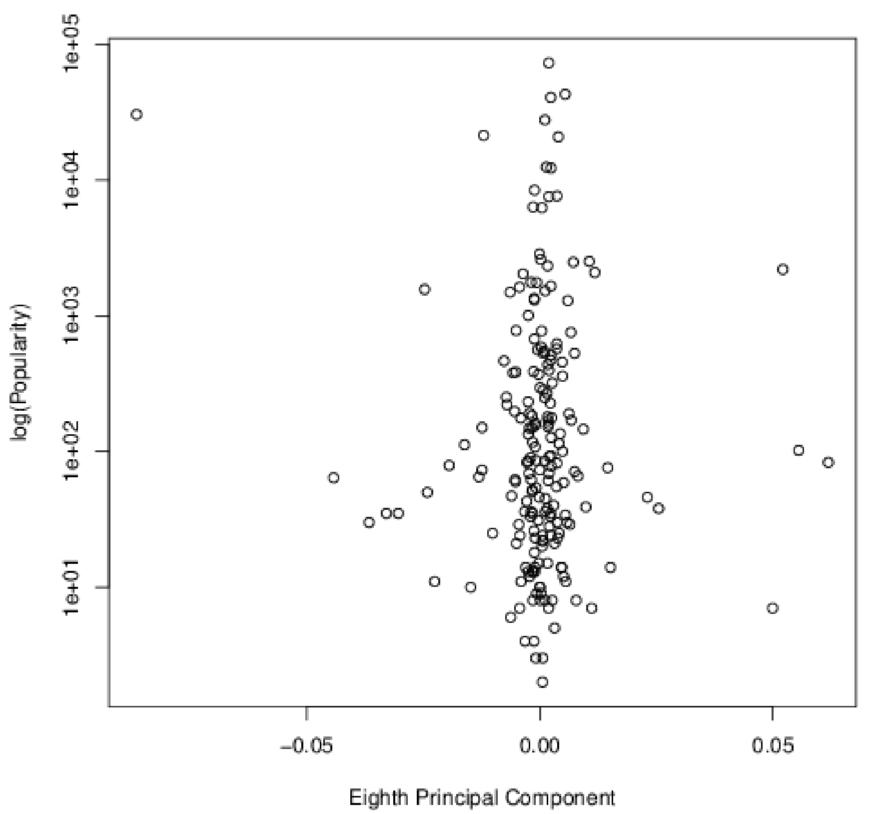


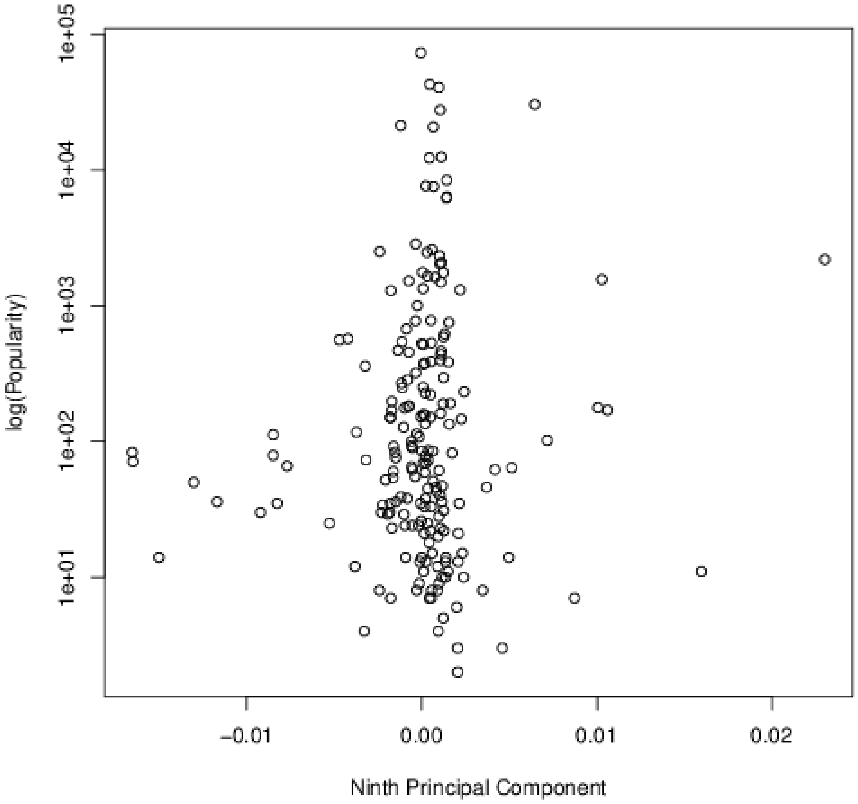
1e+05
1e+04
1e+03
1e+02
1e+01
log(Popularity)
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
Fifth Principal Component



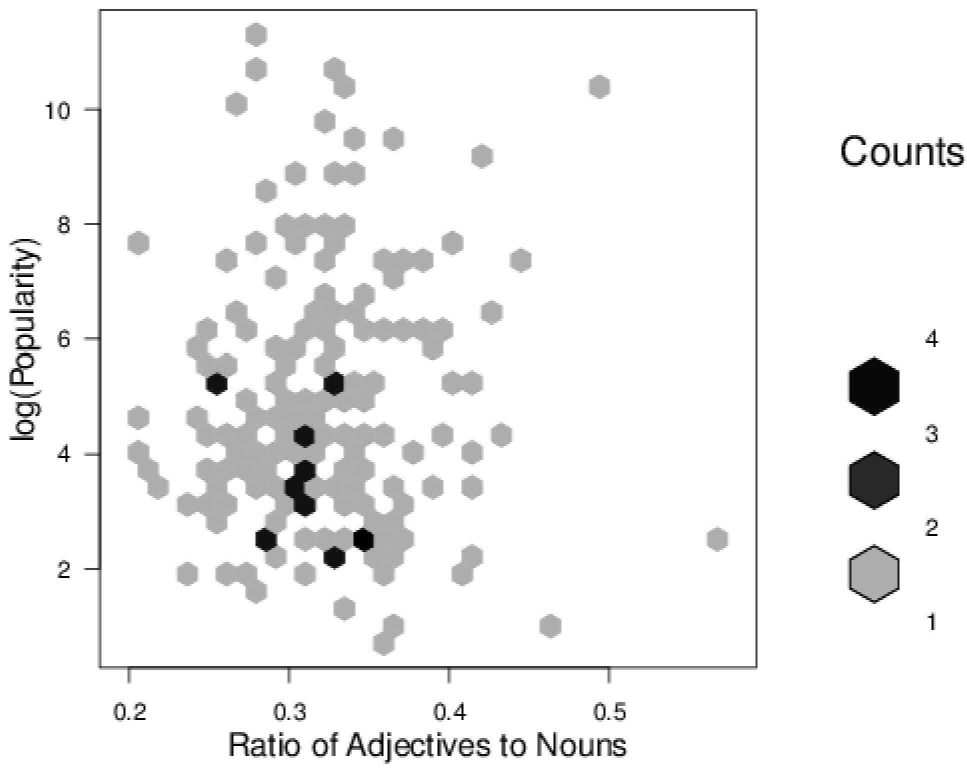
1e+05
1e+04
1e+03
1e+02
1e+01
log(Popularity)
−0.1
0.0
0.1
0.2
Sixth Principal Component

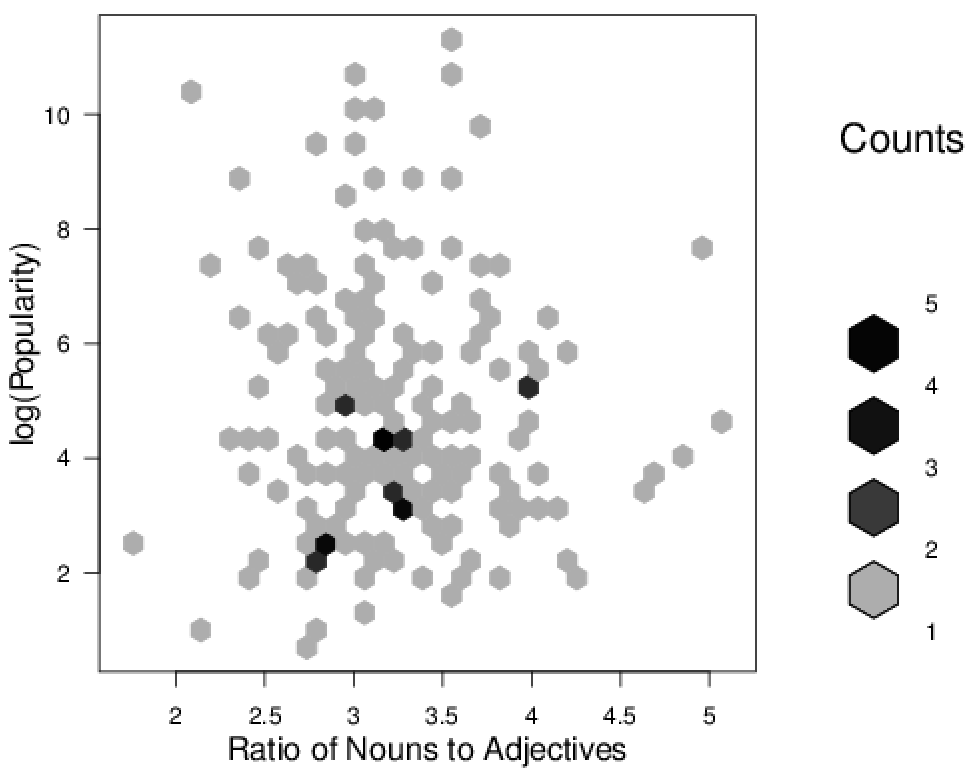

1e+05
1e+04
1e+03
1e+02
1e+01
log(Popularity)
−0.10
−0.05
0.00
0.05
0.10
Seventh Principal Component

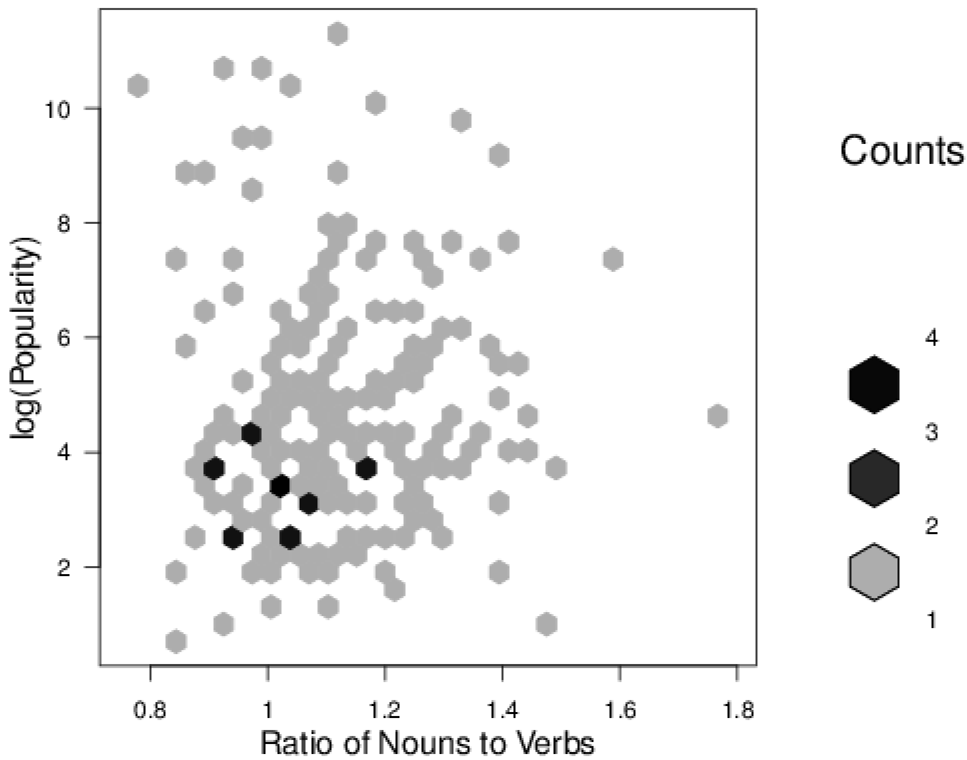

1e+05
1e+04
1e+03
1e+02
1e+01
log(Popularity)
−0.05
0.00
0.05
Eighth Principal Component

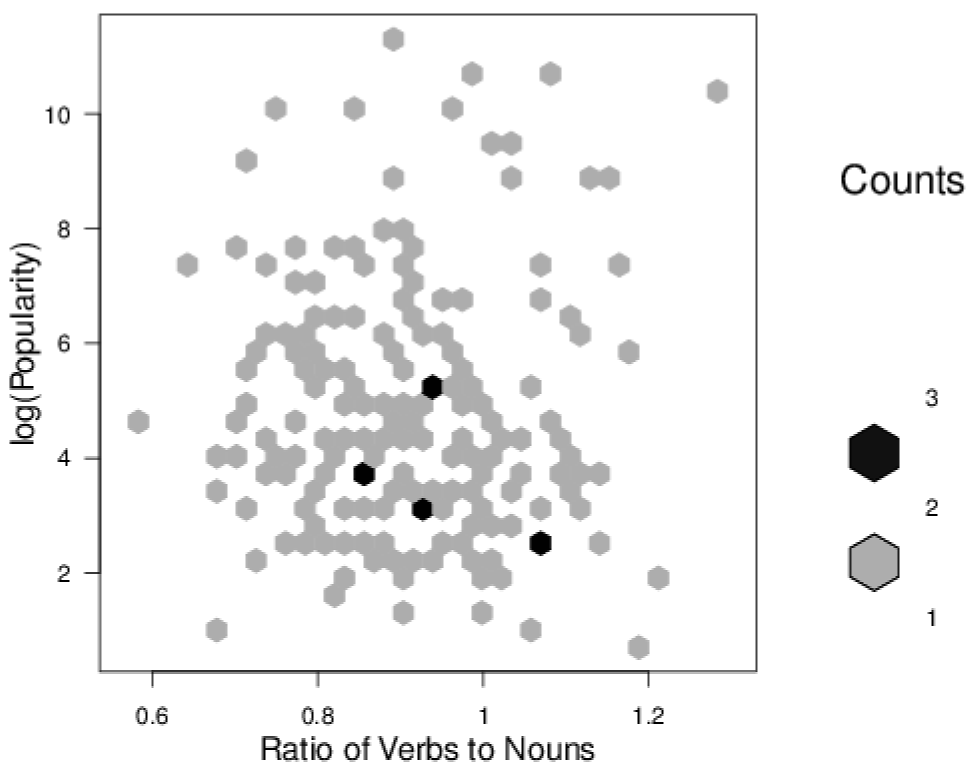

1e+05
1e+04
1e+03
1e+02
1e+01
log(Popularity)
−0.01
0.00
0.01
0.02
Ninth Principal Component

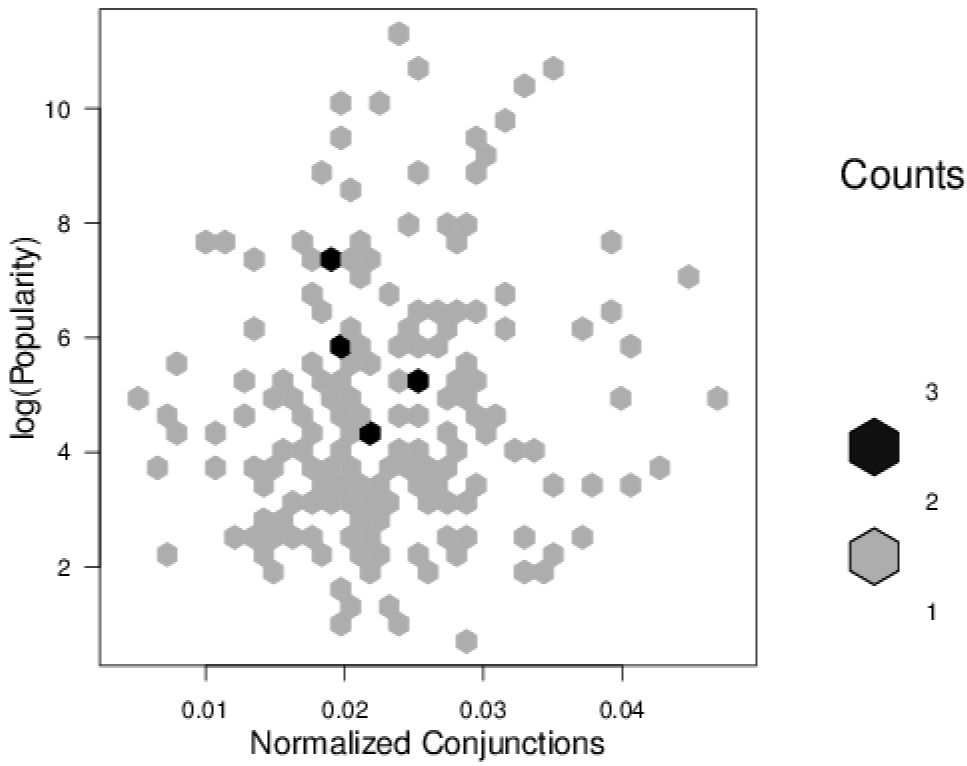
Appendix 7. Additional Plots and Histograms.

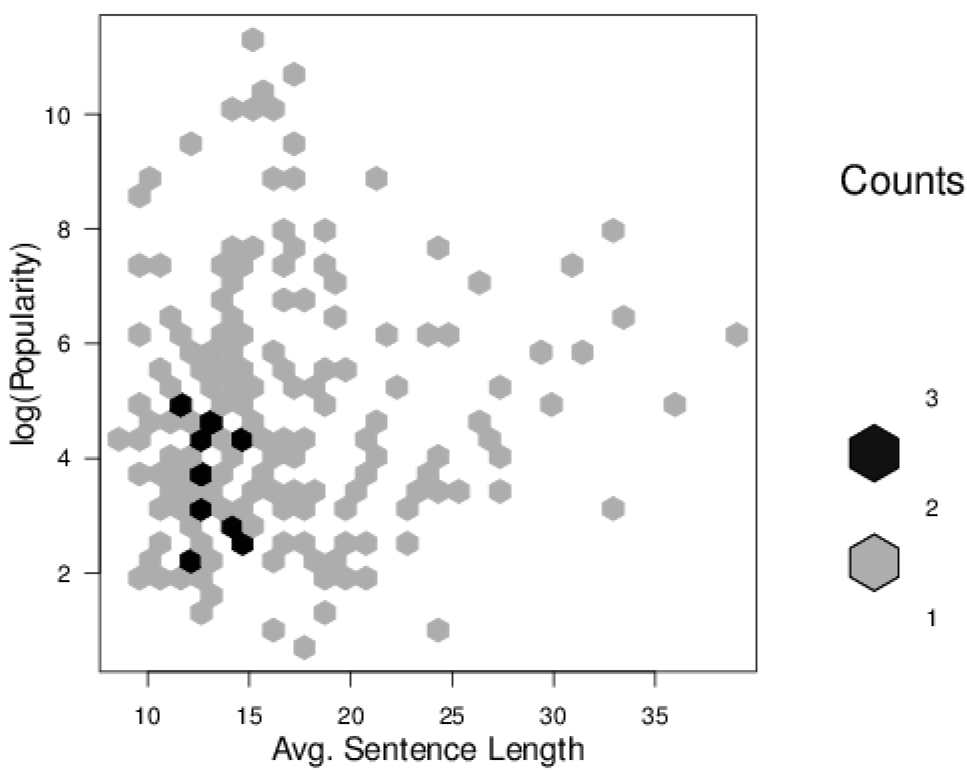


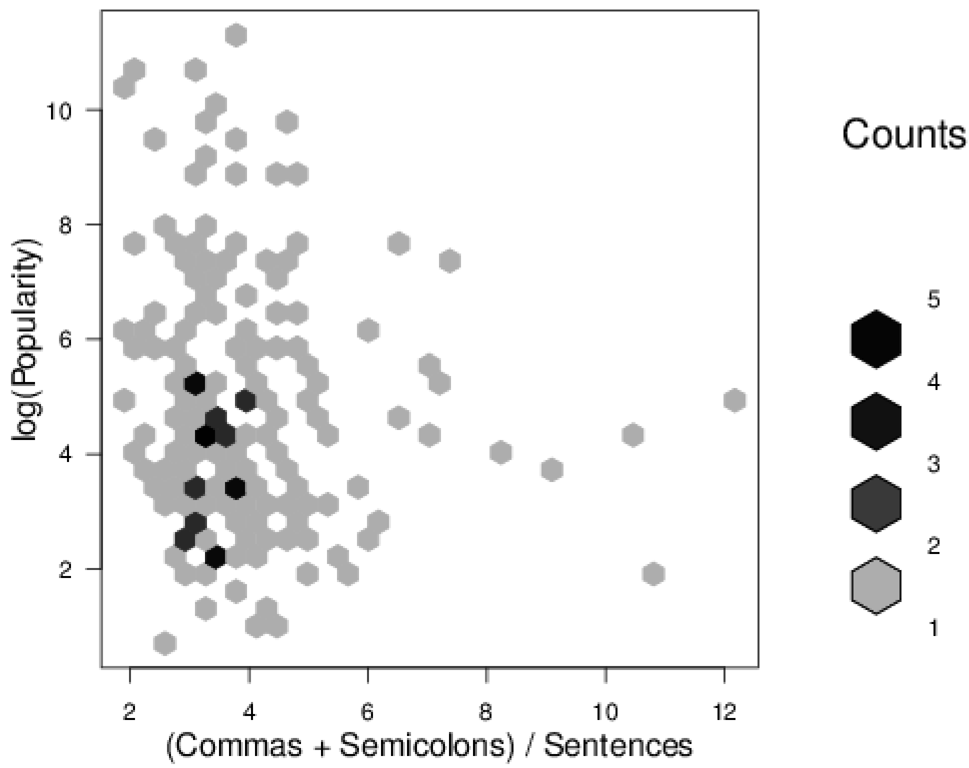



log(Popularity)
Ratio of Nouns to Verbs
Counts
4
3
2
1

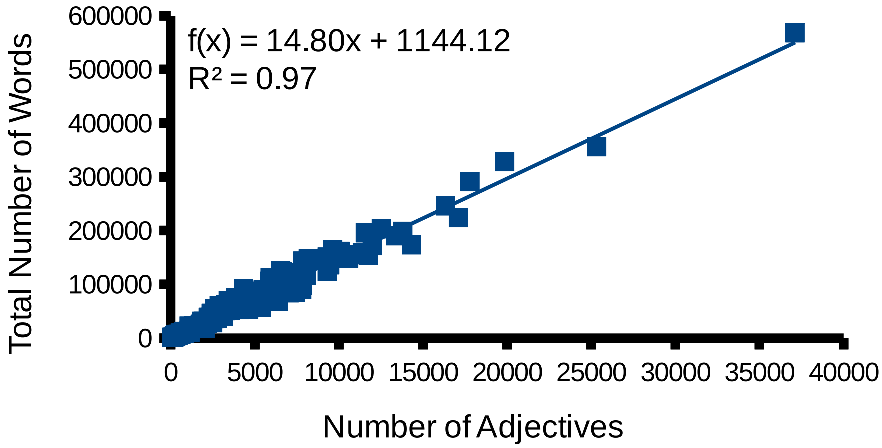

log(Popularity)
Ratio of Verbs to Nouns
Counts
3
2
1


log(Popularity)
10
8
6
4
2
0.01
0.02
0.03
0.04
Normalized Conjunctions
Counts
3
2
1

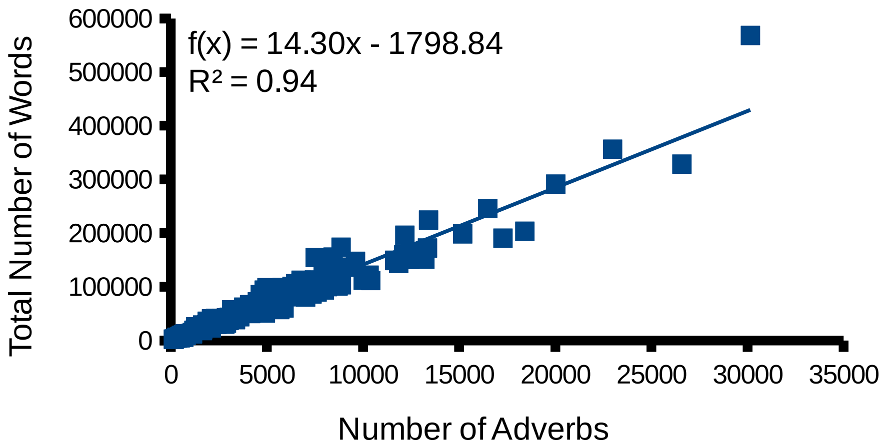

log(Popularity)
10
8
6
4
2
10
15
20
25
30
35
Avg. Sentence Length
Counts
3
2
1



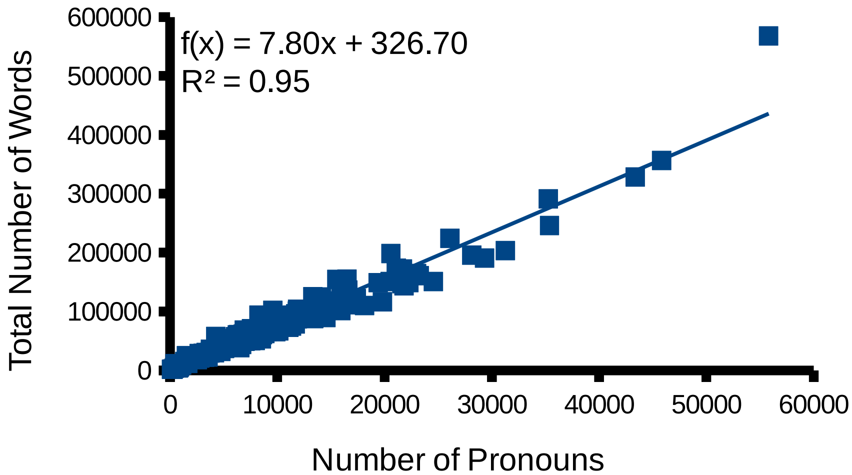
log(Popularity)
10
8
6
4
2
2
4
6
8
10
12
(Commas + Semicolons) / Sentences
Counts
5
4
3
2
1



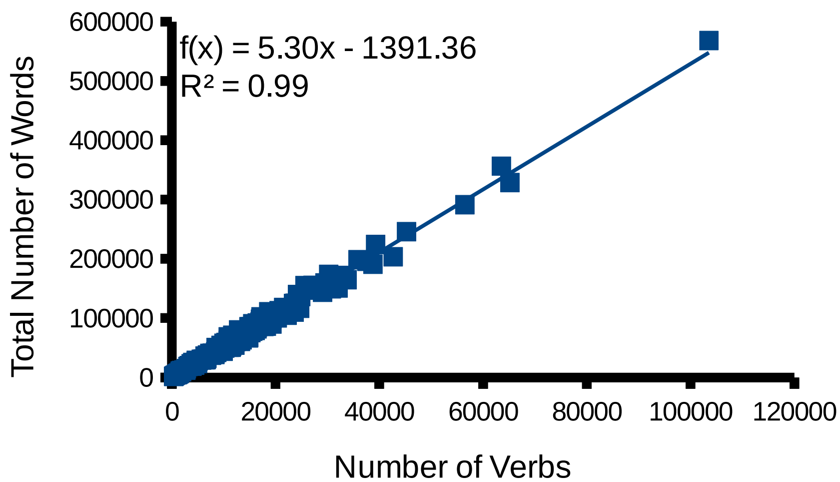
f(x) = 14.80x + 1144.12
R² = 0.97
Total Number of Words
Number of Adjectives
600000
500000
400000
300000
200000
100000
0
0
5000
10000
15000
20000
25000
30000
35000
40000



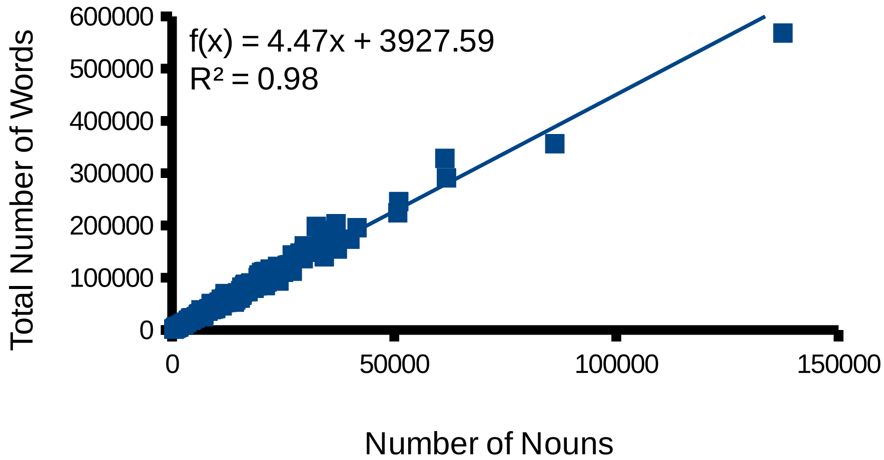
f(x) = 14.30x - 1798.84
R² = 0.94
Total Number of Words
Number of Adverbs
600000
500000
400000
300000
200000
100000
0
0
5000
10000
15000
20000
25000
30000
35000



f(x) = 7.80x + 326.70
R² = 0.95
Total Number of Words
Number of Pronouns
0
100000
200000
300000
400000
500000
600000
0
10000
20000
30000
40000
50000
60000



f(x) = 5.30x - 1391.36
R² = 0.99
Total Number of Words
Number of Verbs
0
100000
200000
300000
400000
500000
600000
0
20000
40000
60000
80000
100000
120000


f(x) = 4.47x + 3927.59
R² = 0.98
600000
500000
400000
300000
200000
100000
0
0
50000
100000
150000
Total Number of Words
Number of Nouns